(6) Datacomputer and SIP Operations: Quarterly Technical Report, July through September 1977

(12)

Technical Report
CCA-77-13
October 30, 1977

Donald E. Eastlake III

Computer Corporation of America

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

(6) Datacomputer and SIP Operations.

(9) Quarterly Technical Report.

Jul[redacted]-Sep[redacted]77,

(10) Donald E. Eastlake, III

(11) 30 Oct 77

(12) 79p.

(14) CCA-77-13

(15) This research was supported by the Defense Advanced Research Projects Agency (ARPA Order [redacted] 3346) No. MDA903-77-C-0183. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

387 285

Table of Contents

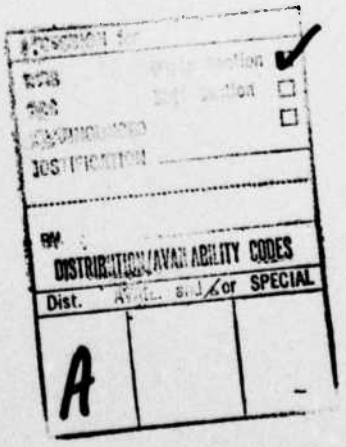


Table of Figures

Project Staff Members

Donald E. Eastlake, III

Jerry Farrell

Stephen A. Fox

David Kramlich

Matthew Maltzman

Gerald E. Maple

David W. Shipman

Steven A. Zimmerman

1. Summary

This report describes our work during the third quarter of 1977 in providing very large on-line data storage and retrieval services over the Arpanet to support seismic data activity and general use. Use of these services continued to grow in the Arpanet community and additional applications are under active investigation.

These services are provided by CCA through the Datacomputer, a system designed to allow convenient and timely access to large on-line databases for multiple remote users communicating over a network. This quarter the Version 4 Datacomputer, with several improvements designed to improve TBM utilization and Datacomputer accessibility, became the operational Datacomputer.

CCA coordinates and assists the seismic and Arpanet communities in utilizing the unique facilities of the Datacomputer so that the greatest benefit may be realized from them. In the seismic field this quarter, the foundation was laid for construction of the Signal Waveform File which will contain the most important of the seismic data that may be sent to the Datacomputer.

Particular assistance was given to Lincoln Laboratories Applied Seismology Group and Texas Instruments (an SDAC contractor). In the general user community, CCA is working with Argonne National Laboratory to determine the suitability of the Datacomputer for the storage of weather data and possible use by other ERDA laboratories. A growing number of systems around the Arpanet are successfully making use of the Datacomputer in a background mode and new uses, such as the storage of facsimile images by the University of London facsimile project, are being demonstrated.

The seismic application is the largest user of the Datacomputer in terms of amount of data stored, complexity, and bandwidth. It sends much of its data to CCA in real time. This real time data stream is fielded at CCA by a small reliable dedicated processor, called the Seismic Input Processor, or SIP, which periodically forwards the data to the Datacomputer. A new SIP communications protocol and numerous improvements in it and its monitor program, STRUGL, were made last quarter. Final improvements to meet all known requirements are now complete. New manuals on the SIP and STRUGL have been appended to this report.

Datacomputer up time declined this quarter due to hardware problems with the Ampex Tera-Bit Memory system controllers as described in Section 4 below.

Although only maintenance and operations of the Datacomputer are included in this contract, active development work for other Datacomputer sites is being conducted by a different group at CCA in support of the ARPA ACCAT project. Some of the fruits of this separate development effort have been made available to users of the TBM based CCA Datacomputer.

The paper Tertiary Memory Access and Performance in the Datacomputer, which appeared as Appendix B to last quarters technical report, was presented on October 7th in Tokyo, Japan, at the Third International Conference on Very Large Data Bases. (This paper is available from CCA as Technical Report CCA-77-11.)

2. The Datacomputer

Below, a general description is given of the Datacomputer followed by an exposition of the enhancements incorporated in the Version 4 Datacomputer which became operational September 25th, 1977.

2.1 General Description

This subsection is a brief general description of the structure of the Datacomputer which is intended to provide the context for the rest of the report. Persons already familiar with the Datacomputer may skip over it. Persons desiring a more detailed description than presented here are referred to the final Semi-Annual Technical Report for the Datacomputer Project, contract number MDA903-74-C-0225, covering July through December 1976, and to the two papers, entitled Use of the Datacomputer in the Vela Seismological Network and Tertiary Memory Access and Performance in the Datacomputer, which were appended to the April - June

1977 Quarterly Technical Report for this contract. These two papers are also available from Computer Corporation of America as Technical Reports CCA-77-08 and CCA-77-11 respectively.

The intended Datacomputer user is a program running on a Arpanet host remote from the Datacomputer. This program calls the Datacomputer over the network and establishes a pair of standard uni-directional 8 bit byte network connections. The user program then proceeds to send Datalanguage over one connection while the Datacomputer replies on the other. Datalanguage is a uniform high level language which gives the user a hierarchically structured view of his data.

Actually, the Datacomputer sends a "reply" to prompt the user program whenever the Datacomputer is ready to accept another line of Datalanguage. Similarly, the Datacomputer keeps the user program abreast of the progress of its requests with various synchronization, error, and success messages and comments. All replies have a fixed format which is designed for easy parsing by the user program. The reply begins with a unique number quickly identifying certain important messages. The remainder of a reply provides a human-readable version of the message to be conveyed. In the case of

serious errors, to assure resynchronization with the user program, the Datacomputer enters a mode where it rejects all messages from the user, giving an appropriate reply each time, until the user program sends a special message to clear this condition.

Data as well as commands and replies can be transmitted between the user program and the Datacomputer over these connections but certain characters are prohibited and the connections are fixed at an 8 bit bytesize. More general data transfers can be done by using a separate data connection.

The requests, replies, and data for a particular user program are handled by the half of the Datacomputer known as RH or the request handler. RH handles the parsing of the user commands, which are in datalanguage, and synthesis of replies. For more complex commands, RH takes the user's requests and the data descriptions stored in the Datacomputer and compiles them, in several stages, into code to execute the request.

Data descriptions in the Datacomputer are associated with each file of data. Data descriptions are also provided for data streams to be received from or transmitted into the Arpanet. These streams are known as ports. Descriptions are set by the user when a file

or port is created. Datacomputer data descriptions provide for hierarchical arrangements of structures of diverse data elements and repetitive lists. Many data types and data formatting alternatives are provided. This is important in ensuring that the Datacomputer can communicate efficiently with its diverse class of remote user computers.

RH accomplishes most of its activities by calling on routines in the other half of the Datacomputer, known as SV or services. SV is a pseudo-operating system in which RH runs. It is SV that actually has custody of the Datacomputer's multilevel hierarchical directory tree and enforces the extensive protection mechanisms of the system. SV is the part of the Datacomputer that ultimately communicates with storage devices and retrieves or stores bits. It views data as a relatively simple collection of areas subdivided into fixed size "pages" of data. All of the finer data description mechanism is in RH.

A special subpart of SV, called SDAX, moves active data from slower tertiary memory to faster secondary disk storage and moves it back when secondary storage is crowded. The CCA Datacomputer uses an Ampex TBM, described in Section 4 below, for tertiary storage.

SDAX keeps track of where various copies of each file data page are. It also ensures data consistency by preventing updates of a file that are not successfully completed from affecting the original file. SDAX allows multiple readers and a single updater of a file among these Datacomputer subjobs. Each reader sees a consistent version of the file including only updates that were complete when they opened the file.

At any one time, there are multiple copies of the RH-SV pair serving users over the network as shown in Figure 2.1. This actually means multiple copies of their variable area as they are implemented in reentrant code. The RH-SV pairs are called subjobs as they are all part of one job, under a monitor process, running in our modified TENEX operating system.

Figure 2.1

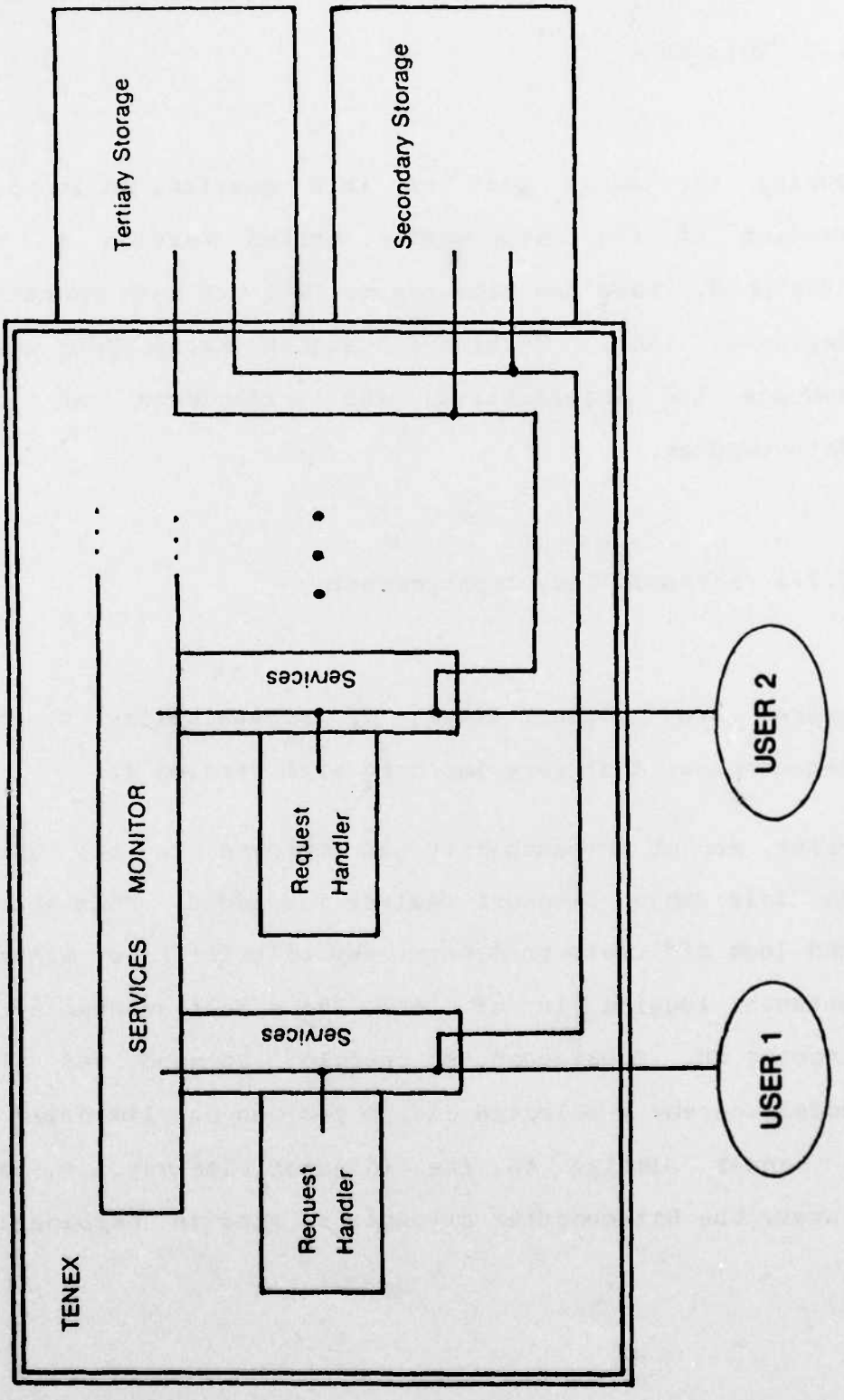


Figure 1. Software View of the Datacomputer

2.2 Version 4

During the early part of this quarter, an improved version of the Datacomputer called Version 4 was developed. tHis new Datacomputer was put into operation September 25th. Version 4 features improvements which enhance the accessibility and efficiency of the Datacomputer.

2.2.1 Accessibility Improvements

There are three levels of accessibility to the Datacomputer that were improved with Version 4.

First, subjob accessibility was improved in two ways. An idle subjob time-out feature was added. This aborts and logs off users that have been idle for five minutes without logging in or idle for a half an hour after logging in. A Datacomputer operator command was also added whereby a selected user's job can be eliminated in a manner similar to the idle job time-out. In both cases, the Datacomputer attempts to send an explanatory

message before closing the network connections to the user. Another Datacomputer operator command was added whereby the operator can send arbitrary messages to one or more selected subjobs.

At a second level, the accessibility of data was enhanced when preventive maintenance or hardware problems limit access to the TBM. This was done through improvements in the device handling routines in the Datacomputer. Facilities were added for suspending TBM operations on one or more drives in a controlled fashion. While operations on a drive are suspended, data recently read from that drive will still be available on disk and data destined for that TBM drive can be written to disk up to the limits of available disk space. All Datacomputer directory information can be accessed and modified regardless of any suspension of TBM activities. If a user program makes a request that references data which is only on TBM, a distinctive message is sent to the user and their job is suspended. The job will be automatically resumed when the referenced TBM drive is enabled by the Datacomputer operator. Alternatively, the user can interrupt out of the suspended state to make other requests.

This suspension mechanism was chosen over the alternative of a new error return so that no modifications would be necessary to the automatic programs around the network that make us of the Datacomputer in a background mode. If data they wish to access is temporarily unavailable, they will simply be suspended and later resumed. The idle job time-out does not run for jobs suspended on TBM operations.

Finally, the third level of TBM accessing improvement was the implementation of the TBM Read Recover feature. If there are problems in reading from TBM tape, the Datacomputer invokes this feature which automatically tries a variety of recovery techniques. If the Read Recover operation succeeds, the Datacomputer takes care that the recovered data is ultimately written back on TBM tape correctly.

2.2.2 Efficiency Improvements

The Version 4 Datacomputer contains efficiency improvements in the writing of TBM tape and in request processing.

Reliable writing on TBM tape requires three steps:

1. erasing the block to be written;
2. writing the new data; and
3. reading the newly writen data to verify that it has been written correctly.

This is a slow operation because of the need to repeatedly start-up and stop the tape drive to traverse the same area. This process has been made more efficient for those (very frequent) cases where a sequence of contiguous blocks are being written. The new technique involves performing each of the steps (erase, write, read-verify) for the entire sequence at once rather than handling the blocks one at a time.

The processing of requests in the Datacomputer was made more efficient in two ways. First, a pre-compiled

request feature was added. This feature enables a user to store a request under a name and to execute the same datalanguage any number of times in the same session without compilation overhead. This pre-compiled request feature was done as part of the ARPA IPTO-ACCAT project and was also made available on the TBM Datacomputer at CCA. The second way request efficiency was increased was through improvements in the lock routines used to interlock critical areas between Datacomputer subjobs.

2.2.3 Virtual Memory

Some difficulty was encountered in finding enough room in the Datacomputer's virtual memory space for the above improvements. This problem was solved by modifying the Datacomputer debugging program. The debugger was modified to swap the Datacomputer symbol table in and out of memory, thus avoiding the storage cost of this data when it is not required.

3. The User Community

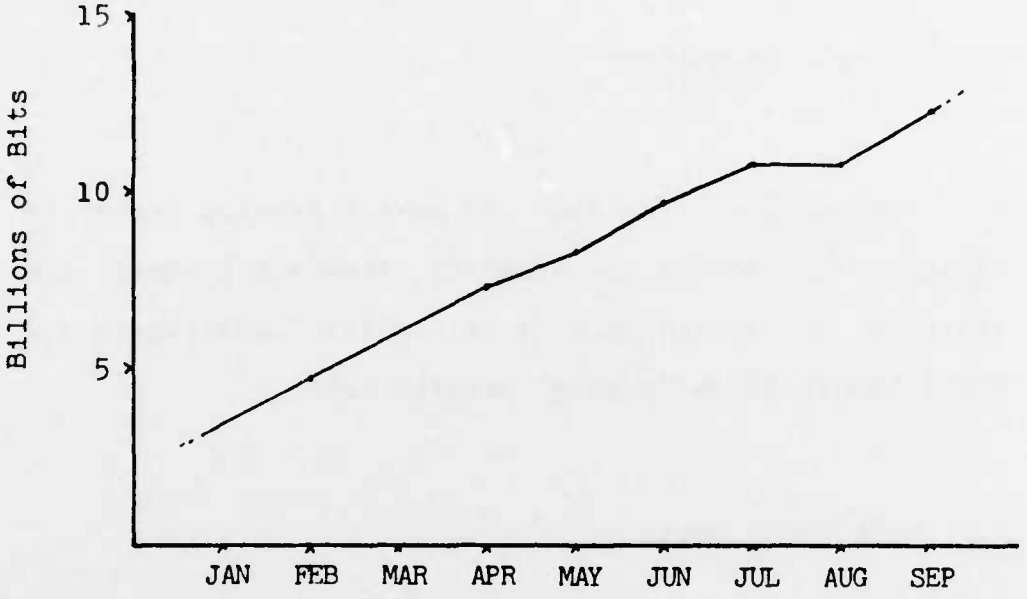
The Datacomputer continue to have a growing number of diverse users around the Arpanet. Figure 3.1 shows the increase in non-seismic data on-line and Figure 3.2 shows the increase in total seismic data.

3.1 Background Usage

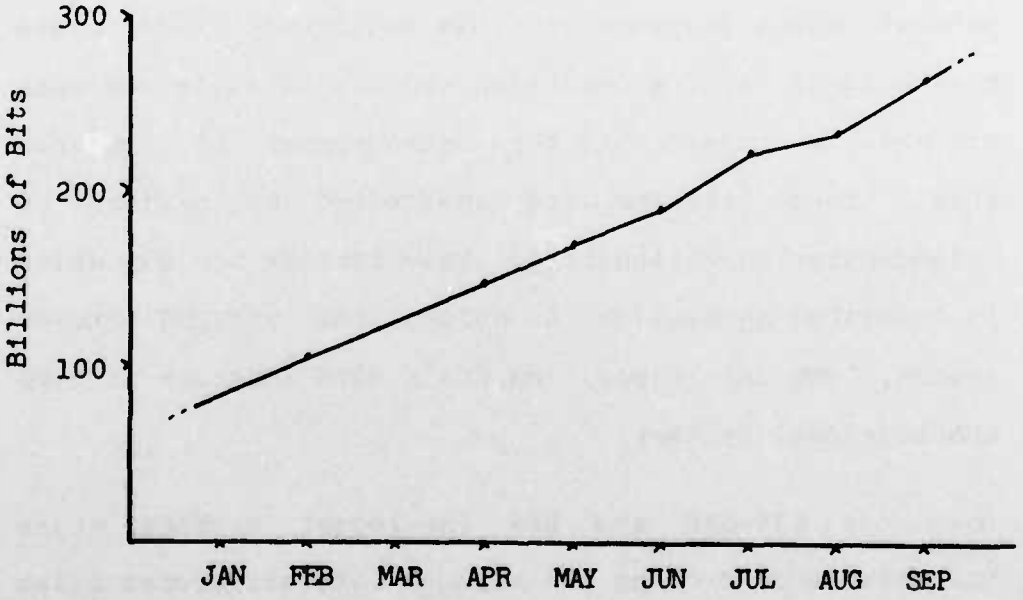
There are now four systems on the Arpanet which automatically store data in the Datacomputer. In general, these programs run in a background mode where transactions are accumulated and periodically attempts are made to connect with the Datacomputer to complete them. These systems are unaffected by periods of Datacomputer unavailability. They include the SIP which is described in Section 5 below, the MIT-DMS survey system, BBN IMP-logger, and CCA'S MARS (Message Archive and Retrieval System).

Both the MIT-DMS and BBN IMP-logger systems store information concerning the Arpanet into structured files

Non-Seismic Data On-Line Figure 3.1



Seismic Data Figure 3.2



in the Datacomputer. The MIT-DMS survey system periodically surveys the status and connect response time of the hosts on the network. All of this survey data for several years back is available on-line in the Datacomputer. The BBN IMP-logger system accumulates data on IMP status and unusual events in the network communications backbone. This information is now available for over a year in the Datacomputer.

CCA's Message Archiving and Retrieval System is somewhat different. It periodically stores into the Datacomputer all of the Arpanet mail arriving at a special mail box at CCA-TENEX. These messages are stored into relatively complex files set up to utilize the Datacomputer's automatic indexing facilities. Messages can later be efficiently retrieved by the other half of the MARS system based on a variety of criteria. (The MARS program was developed under contract No. N00014-76-C-0991 and is described in CCA-77-10, Very Large Databases: Final Technical Report.)

3.2 Seismic User Community

CCA coordinates and assists the seismic user community of the Datacomputer.

This quarter the basic foundation for the seismic Signal Waveform File (SWF) was laid at a meeting held August 26th at Lincoln Laboratories. Dr. James Rothnie and Donald Eastlake attended this conference for CCA and representatives of ARPA-NMRO, Vela Seismological Center, Lincoln Laboratories Applied Seismology Group (LL-ASG), and Seismic Data Analysis Center (SDAC) were present. The SWF will contain the seismic data that is believed to be associated with particular seismic events. This is the most important of the seismic data. Its lesser volume, in comparison to the raw seismic data, will enable it to be kept on line longer.

CCA also assisted Texas Instruments, as SDAC contractor, and LL-ASG is designing and using datalanguage to deal with seismic data.

Approximately every six weeks, the short and long period seismic data being sent to CCA fills up a TBM tape reel.

CCA continues to manage the allocation of these files to tapes and the migration of older tapes off line. During this quarter off line tapes were temporarily mounted on a few occasions at the request of LL-ASG.

3.3 General Usage

General use of the Datacomputer continued to spread over the Arpanet and additional applications are under investigation.

Jerry Farrell and David Kramlich of CCA visited Argonne National Laboratory (ANL) and are assisting ANL in loading a weather database into the Datacomputer. This will be used to determine the suitability of the Datacomputer for this application by ANL and other ERDA laboratories.

Use of the Datacomputer for archival storage of files is growing from the TENEX, ITS, TOPS-10, and MULTICS systems on the Arpanet. An improved archival format that will be efficiently usable by UNIX and other non-36 bit systems is being designed.

The University of London Arpanet project has found the Datacomputer useful in the storage and retrieval of

facsimile images. Their annual report issued in February 1977 says of the Datacomputer "... it does contain software and hardware ideal for archival storage. Thus it is an excellent vehicle for our experiments."

The automatic Datacomputer status server program at CCA has been enhanced. This program listens on socket 703 and gives users a computed message concerning Datacomputer status. It now gives a clear indication if CCA-TENEX is aware of problems with the TBM controllers, CalComp disks, or the SA-10 data channel.

4. TBM Mass Storage System

After a brief overview of the Ampex TBM, this section discusses its recent reliability problems and some improvements that have been made in the CCA TBM test utility, TBMUTL.

4.1 General Descripion

The CCA Datacomputer has been equipped with the first public installation of the Ampex Tera-Bit Memory (TBM) System. This device uses video tape technology to achieve a maximum on-line capacity of around 3 trillion bits. Maximum seek time to any bit is 45 seconds. Maximum data transfer rate is 5.3 million bits per second.

The TBM at CCA is equipped with two dual tape transport modules so at most four tapes, or about 176 billion bits (equivalent to 220 IBM type 3330 disk packs) can be available on-line. All equipment between the transports and the Datacomputer central processor is non-redundant

in the present CCA configuration. There is one transport driver (necessary for a tape to be in motion), one data channel (necessary to encode and decode digital information to and from the broadband signal on tape), one system control processor to coordinate the TBM, and one channel interface unit that connects the TBM system to the Datacomputer's PDP-10 system. The system is diagramed in Figure 4.1. Data is transferred directly between TBM tape and core memory.

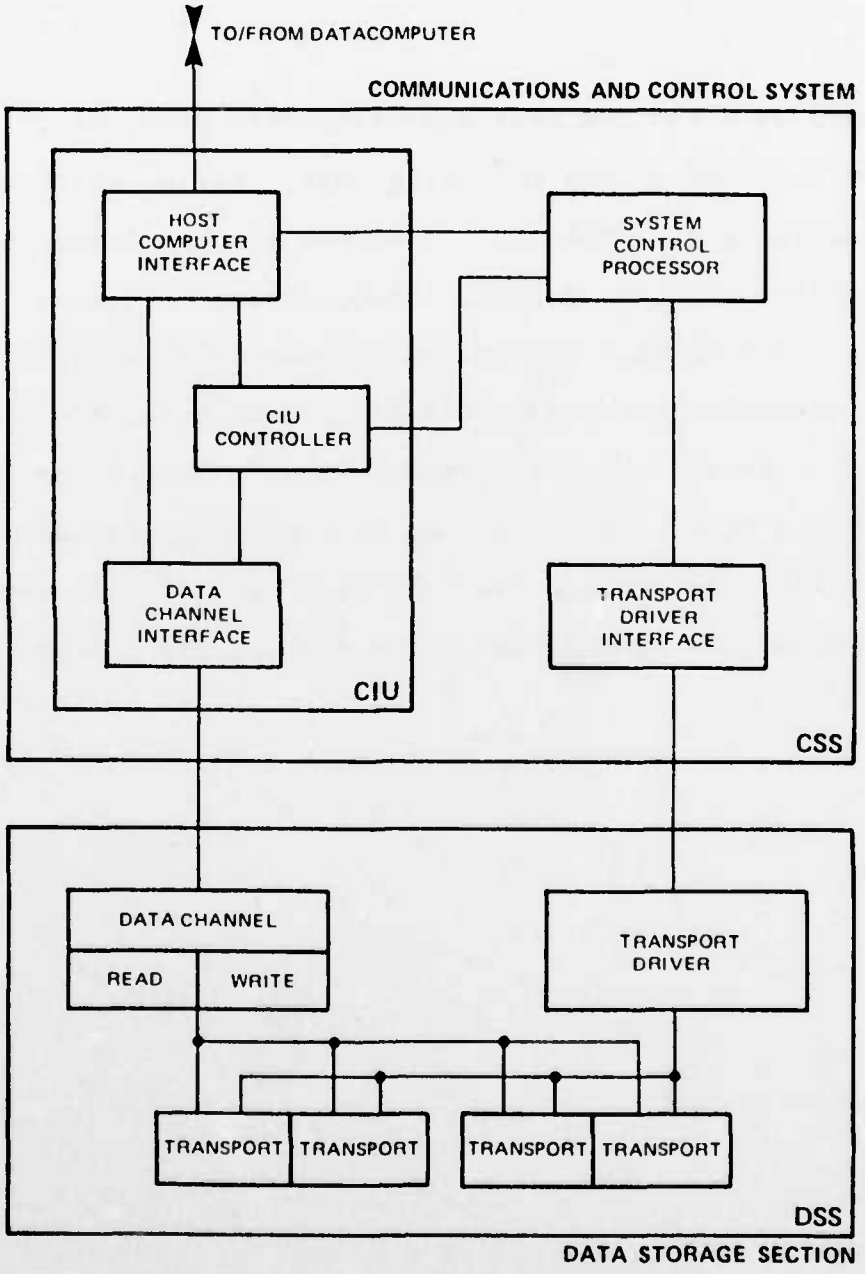
4.2 TBM Reliability

The TBM suffered from renewed reliability problems this quarter. This decreased the availability of the Datacomputer during prime time to 76%.

Almost all of these problems were in the Host Channel Interface unit (HCIF) of the CIU. The HCIF interfaces with the Data Channel Interface, the CIU controller, and the IBM 370 block multiplexor compatible channel to the PDP-10. The CCA HCIF is the first of its kind made by Ampex.

CCA continues to press Ampex for greater reliability and a shorter mean time to repair.

CCA TBM Structure Figure 4.1



4.3 TBMUTL

TBMUTL is CCA's TBM test and diagnostic program. It has been improved in the following ways: (1) a quick test sequence was added for checking out TBM drives after cleaning, (2) a printing suppression feature was introduced to make TBMUTL more usable on slow terminals, (3) commands were added for doing Read Recover and Auto Align operations, (4) TBMUTL was modified so that multiple drive tests could be done more conveniently and (5) a convenient top level command was added to read in and clear the operations counts for all TBM drives.

5. The SIP

The final improvements in the SIP to meet all known requirements were made this quarter. These are given below after a general description of the SIP. New manuals have been prepared for the SIP and its monitor/debugger called STRUGL. These manuals have been appended to this report. The high volume of data flowing to and from the SIP has had significant Arpanet impact but appears to be working smoothly now.

5.1 General Description

The Seismic Input Processor (SIP) matches the real time continuous stream of seismic array data arriving at CCA from the CCP to the Datacomputer. It is a small dedicated system implemented on a DEC PDP-40 computer with 2 RP04 disk drives and an Arpanet interface.

As the real time data being received is stored on disk it is also reformatted and prepared for periodic bursting to the Datacomputer. Up to 44 hours of data

can be buffered on disk at the present bandwidth without manual intervention.

The SIP also provides for operator communication between itself and the CCP in Alexandria, Virginia, and sends messages to the CCP when data has been properly filed in the Datacomputer.

5.2 New SIP Version

A new version of the SIP became operational late this quarter. An operator manual for it appears as Appendix A to this report. (A new version of STRUGL, the SIP debugger and basic monitor system has also be developed and installed. It is documented in Appendix B.) Among the improvements and features now in the SIP are the following:

1. Up to eight seismic data sites can be accommodated with only directory changes. (No program changes will be necessary.)

2. Disk packs can be changed while the SIP is running and the storage and retrieval of directories is handled automatically.

3. The SIP avoids using the Datacomputer when the Datacomputer is heavily loaded unless the SIP disks are nearing their capacity.

4. The SIP has the full retransmission logic of the new CCP<->SIP protocol implemented. However, retransmission to the CCP is presently inhibited as the CCP does not yet send acknowledges.

5. Whenever the local CCA IMP indicates it is going down by a message to the SIP, the SIP sends a computed "operator" message to the CCP to inform the CCP operator of this.

6. When the SIP senses a power failure it attempts to send a message to the CCP and CCA IMP indicating it is going down and why.

5.3 Arpanet Considerations

Communications difficulties had been experienced on the CCP<->SIP path and particularly in the CCA IMP due to the high bandwidths being transmitted. As a result, a new CCP<->SIP protocol was designed that minimized the number of messages sent between them and the previous CCA IMP was replaced by a PLURIBUS IMP early this quarter.

These changes appear to have solved the problem at current bandwidths. There has been no network congestion noticeable to CCA since they were implemented.

APPENDIX A:

SIP Operator's Manual

by Steven A. Zimmerman

Computer Corporation of America

15 October 1977

INTRODUCTION

This manual contains useful information for anyone who is called on to be a SIP "operator". The SIP's purpose is to receive seismic data over the Arpanet in real time from the CCP (Communications and Control Processor) at the Seismic Data Analysis Center in Alexandria, Virginia. This data is reformatted and buffered on the SIP's disk and periodically burst through the CCA-IMP into the Datacomputer.

The SIP runs under a debugger/micro-operating system known as STRUGL (SIP's Total RUG Language). This manual assumes a working knowledge of STRUGL; complete information on its use can be found in the STRUGL User's Guide. For all STRUGL commands in this manual, an altmode is indicated by a dollar sign ($) and a control character is indicated by an uparrow (^) preceding the character.

You can usually tell if the SIP is dead by the fact that no part of the status display is being updated (it's all right if only the first few lines are being updated). The SIP will normally have hit a BUGHLT/BUGCHK and be in STRUGL typing out periodic plantive bells or have halted (light on console labeled "console" on). In cases of difficulty, you should try contacting one of the following people:

ID	Home Phone - Name
Z	396-7811 - Steve Zimmerman
DEE	244-2679 - Donald Eastlake (home)
DEE	354-7519 - Donald Eastlake (beeper)
NATG	776-3731 - Nat Goodman
GEM	395-9406 - Gerry Maple

Other telephone numbers that may be of use:

NCC	661-0100 - Network Control Center
SDAC	703-836-3882 - Seismic Data Analysis Center
VSC	703-325-8126 - Vela Seismological Center

TABLE OF CONTENTS

THE STATUS DISPLAY

The status display is designed to show what is going on and what has recently happened in the SIP. Its contents are described below. Occasional inconsistencies may appear in the display because the screen is being updated over a length of time during which the information can change.

Line 1

From left to right, line one has (a) the current SIP clock (GMT, followed sometimes by a plus or minus and a diddle factor which has to do with relative drift of the SIP and CCP clocks), (b) a character that is incremented once each time the core checker checks all of core, (c) the time and description of the last "significant event" such as IMP flaps, power fails, or unusual messages received, and (d) the first two letters of the name of any site from which the SIP receives data but does not recognize.

Lines 2-4 - Left and Center

This area gives information on seismic data actually being received. Each site code is followed by the time code of the last data correctly received from that site. The time code is in the form year (i.e., 077 for 1977) followed by day of year and time, GMT. An "s" and/or "l" usually appears before the time code to indicate correct receipt of short or long term data. If the SIP's clock is out of synch with the data, the time code will be preceded by "?T". If the SIP is zeroing a slot for this data, the time code will be preceded by "zs". If current data is being received, the most recent time code should be about 30 seconds behind the SIP clock (which is kept 30 seconds ahead).

Line 2 - Right

This line shows the allocation of core buffers to various tasks. A period indicates an idle buffer. Other characters indicate how the buffer is being used; however, in many cases two, four, or even eight adjacent buffers are assigned to one use and in these cases the use letter appears only on the leftmost buffer with the remainder being labeled 2 or 234 as the case may be. Some letters and corresponding uses are as follows:

? NCP buffers
a Acknowledge buffer
c CCP message
d DC job work area
i IMP input
l Lasa data from CCP
n Norsar data
o IMP output
r Retransmission buffer
t Type in/out
u Directory update buffer
v Volume validate buffer for system copy

Lines 3-4 - Right

This area shows the situation with regard to status messages. Line 3 gives the number and time code of the last status message succesfully retransmitted to the Datacomputer. Line 4 gives the same information for the last status message received from the CCP. The number preceding the plus is the number of status messages successfully sent to the Datacomputer, while the number preceding the minus is the number which have been received from the CCP. In addition, the minus sign is followed by a letter for the SIP pack on which status messages are currently being stored.

Lines 5-20

These lines show, in the left and central parts of the screen, the state of various internal queues in the SIP. The columns are, from left to right, Q0 Q1 Q2 Q3 BFW DSK ALARM. Q3 is the highest priority and Q0 the lowest.

Each entry has the form: T1234Q where T is the task's ID, 1234 is the task'c PC in octal with digit 5 omitted, and Q is the ID letter of who queued the task. An extra "<" shows the "next in" pointer for queueing tasks and a ">" indicates "next out" for tasks to be run. A "+" indicates a live unrun task and a "-" indicates a dequeued unrun task (can only happen in the Alarm queue).

At the right edge of the screen some counters are listed. These may change from time to time and new ones may be added. The current ones are as follows:

Bad8 - This is a count of the number of bad status messages received from the CCP. It is the sum of the Fun, Old, and Nil counters.

Good - This is a count of how many otherwise good status messages were thrown away because they could not be stored on disk or because the SIP's clocks were not in synch with the CCP's.

fun - This is the number of status messages which arrived from the CCP with a day or year time code of zero, or with a time code that fell outside the current open hour slot(s).

NoDr - This is the number of status messages which had to be thrown away because there were no disk drives available on which to store them.

old - This is the number of status messages which arrived from the CCP with time codes that were too old.

full - This is the number of status messages which had to be thrown away because there was no room to store them on the disks.

nil - This is the number of status messages which arrived from the CCP containing no information.

Bak8 - This is the number of status messages whose time code was earlier than the preceding status message.

Wer8 - This is the number of disk errors encountered while writing status messages to the disk.

out - This is the number of status messages which were written onto the disk out of order.

Line 10 is reserved for the time codes for data from sites 7 and 8, if they ever appear.

Line 11 contains the CCP status indicator; it consists of the

name of the CCP which is currently up (either CCP47 or CCP76) followed by an uparrow. If the CCP is dead, this indicator is blank except for the uparrow.

up - This is the number of hours the SIP has been up since the last power fail, bughalt, or Control C.

Future - This is the number of type Ø messages received from the CCP with time codes more than five minutes into the future. These messages are discarded.

OddCt - This is the number of messages received from the CCP with odd numbers in their length fields.

HTlost - This is the number of heads and tails under two minutes old which had to be flushed because the head or tail buffer was full.

AKlost - This is the number of times the acknowledge buffer had to be flushed because it was too full. Each time this buffer is flushed, 56 acks are lost.

NoHID - This is the number of hours of data sent to the Datacomputer which did not have an ID letter and number somewhere on lines 23-26.

WerØ - This is the number of disk errors encountered while writing type Ø data messages to the disk.

BadWdCount - This is the number of CCP messages which had unreasonable or contradictory word or byte counts in their length fields.

EchoOther - This is the number of messages that the SIP received from hosts other than itself, the CCP, or CCA-TENEX; these messages are simply echoed back to the sending host.

DupMs# - This is the number of messages that the SIP has received from the CCP which are duplicates of messages which it has recently acknowledged. These messages are discarded.

seek - This is the number of overlapped seeks the SIP has performed.

Abort - This is the number of disk requests which were aborted because the requested drive was dead.

ReTry - This is the number of retries attempted after a correctible disk error was encountered.

ECC - This is a count of how many correctible data errors have occured on the SIP's disks.

Line 21

The most recent Datacomputer Datalanguage reply message (usually truncated to fit) is displayed on this line followed by two letters for the pack and slot that was then being sent to the Datacomputer; SC stands for Status Cylinders.

Line 22

The most recent operator message received from the CCP operator, truncated to fit if necessary, is displayed on this line, followed by the last slot that was sent to the Datacomputer. If the CCP operator begins the message with two exclamation points, it will blink on the SIP's display. To stop the blink, send an operator message to the CCP.

Lines 23-26

These lines show the recent history of the SIP in storing CCP data on its disks. The last entries are the most recent and the entire area conisdered as one string is shifted left when new entries are made. Each entry conists of a capital letter for the pack the data is on followed by a letter for the slot on that pack followed by two digits which are the hour represented. Once the data for that hour has been sent to the Datacomputer, the letter representing the pack is shifted to lower case. Gaps in the history represent interruptions in data from the CCP. The last slot in the string (which should be the second slot on Line 26) is the current open slot; its hour number should be the current hour GMT. Following this slot and separated from it by a space are the names of the next slot to be filled on each pack. The number preceding the plus is the total number of hours of data received from the CCP, while the number preceding the minus is the number of hours which remain to be sent to the Datacomputer. The number to the right of the minus is the number of hours of data which the SIP was forced to throw away when it overflowed its disks.

Line 27

The number with an arrow pointing to it in the extreme lower left of the display is the number of site-period hours of data on disk which have yet to be sent to the Datacomputer. If all sites have sent data during a given hour, this number will be incremented by the number of sites times two (for long and short period). The number preceding the double bars is the number of site-period hours which have been lost because the SIP overflowed its disks.

SIP TOGGLE SWITCHES

Switches 15, 14, 13, 12

These switches are useful in causing part of the status display to be updated more rapidly so that it is possible to see what is happening with the data streams from the CCP, buffer allocation, or whatever. If switch 15 is off, the rest of the switches are ignored. See the following table where *** means "does not matter":

15-	14-	13-	12-	switch
off	***	***	***	full screen updated
on-	off	***	***	only top line updated
on-	on-	off	***	only top two lines updated
on-	on-	on-	off	only top four lines updated
on-	on-	on-	on-	only top eight lines updated

Note that switch 12 also enables Control C, and in combination with switch 10 enables Control K. Leaving switch 13 set prevents the SIP from automatically restarting after bughalts. (This is actually a STRUGL function.)

Switch 11

When this switch is on, the NCP, which is the part of the SIP that handles communications with the Datacomputer, prints certain network messages it gets in octal. This is used for low level debugging.

Switch 10

When on, this switch inhibits starting new sessions of seismic data transmisson to the Datacomputer. (Internally, Bit 15 in DCFLAG is set.) If there is a current transmission, it will finish. Sessions normally start two minutes after every other hour or when a Q is typed on the console.

Switches 9 & 1

If toggle switch 1 is on, then toggle switch 9 specifies which disk drive to use (off=zero, on=one) for storing seismic data received from the CCP. If toggle switch 1 is off then switch 9 chooses between two ways of storing data on both disks. If 1 is off and 9 is on, hours alternate between disks. If both 1 and 9 are off, then the disks are used sequentially with the switch done when the next hour slot on a disk would have looped the disk back to its lowest numbered cylinders.

Switch 8

If on, this switch inhibits the start up of any IMP or disk IO activity on power fail and other restarts. It should be used to proceed from IMP or disk breakpoints.

Switch 7

When on, this switch suppresses the printing of "strange messages" and "CCP Bad Sum" system messages. The message type followed by a question mark is still printed out for the bad sum messages, though.

Toggle Switches

Switch 6
If the disk packs become full, the SIP will not throw away old data when this switch is set; it will throw away the new, incoming data instead.

Switch 5
When on, this switch inhibits the printing on the console DECwriter of the heralds announcing operator messages from the CCP operator. It also inhibits printing of the "^Z=SENT" message.

Switch 4
Used for debugging. When on this switch allows one change to come or go in "pure" code without setting off the pure code checker. It is not necessary to set this switch when changes are made from STRUGL.

Switch 3
When on, this switch stops network messages from hosts the SIP does not know about from being echoed back to them. It is useful for inhibiting the running of SIP-whizz programs at random hosts.

Switch 2
When on, this switch causes staging messages to be printed out on the console DECwriter.

Switch 1
See switch 9.

Switch 0
When this switch is on, unacknowledged messages are not retransmitted.

CONSOLE DECWRITER - OUTPUT

Most of the type out on the console terminal is self explanatory. It includes the Datalanguage sent to the Datacomputer and some of the more important replies, operator messages from the CCP operator, various messages concerning network errors and buffering overflows causing loss of data, and other messages. In general console output of a higher priority can appear in the middle of lower priority output.

There are some events that can occur asynchronously and occasionally happen quite frequently that are printed out as one or two characters rather than a whole line. In particular, a letter followed by the digit 8 means that a status message was received by the SIP and stored on the disk pack corresponding to the letter, a digit followed by a question mark means that a bad checksum message was received of a type corresponding to the digit (usualy there is a full line message also), and an M, H, or D followed by a question mark means that the SIP has noticed that its clock differs significantly from the CCP's in either the minutes, hours, or days field. An M, H, or D which is not followed by a question mark means the SIP has just adjusted its clock forward by this amount, while a plus or minus sign indicates that the SIP's clock has been adjusted forward or backward just a little to keep it aligned about 30 seconds ahead of the CCP. A set of square brackets ([]) indicates that the SIP has just finished its IMP initialization routine. If the SIP prints out a slot name followed by an underbar and an exclamation point (such as AA_!), this means that this slot was cleared at the last second and will immediately be used to store data.

Often the SIP will type out a line of the form

SAT24JUL76GMT15:29:46.5 IMP.MSG=abcd (description)

where abcd is a four digit number of which ab is the message type and cd is the subtype. A short description of the message usually follows the message number. Some common message types and subtypes which appear on the printout are:

- 01 Error in Leader - The IMP detected an error in a previous SIP-to-IMP message and had to assume that the leader was garbled. Subtypes:
  - 00 IMP's Error flip-flop set during the first 96 bits of a message.
  - 01 IMP received a message of less than 32 bits.
  - 02 IMP received a message of an illegal Type.
- 02 IMP Going Down - The IMP will transmit this message to the SIP before it voluntarily goes down. The SIP forwards the information in the message to its users from the network. The subtype field contains the number of minutes the IMP expects to be down divided by five. A zero in this field means the IMP doesn't know how long it will be down.
- 04 NOP - The SIP discards this message. It is used during initialization of IMP/SIP communication.
- 06 Dead Host Status - This message usually follows a Type 7

message. Its subtypes are:

- 01 The destination Host is not communicating with the network - it took its ready line down without saying why.
- 02 The destination Host is not communicating with the network - the Host was tardy in taking traffic from the network without saying why.
- 03 The destination Host does not exist to the knowledge of the NCC.
- 04 The IMP software is preventing communication with this Host; this usually indicates IMP software re-initialization at the destination.
- 05 The destination Host is down for scheduled P.M.
- 06 The destination Host is down for scheduled hardware work.
- 07 The destination Host is down for scheduled software work.
- 08 The destination Host is down for emergency restart.
- 09 The destination Host is down because of power outage
- 10 The destination Host is stopped at a software breakpoint.
- 11 The destination Host is down because of a hardware failure.
- 12 The destination Host is not scheduled to be up.
- 15 The destination Host is in the process of coming up.

07 Destination Host or IMP Dead (or unknown) - This message is sent in response to a message for a destination which the IMP cannot reach. The message to the "dead" destination is discarded. Subtypes:

- 00 The destination IMP cannot be reached.
- 01 The destination Host is not up
- 03 Communication with the destination Host is administratively prohibited.

08 Error in Data - The IMP's error flip-flop was set after transmission of the leader of a message but before the end of the message.

09 Incomplete Transmission - The transmission of the named message was incomplete for some reason. An incomplete transmission message is similar to a RFNM, but is a failure indication rather than a success indication. Subtypes:

- 00 The destination Host did not accept the message quickly enough.
- 01 The message was too long (in excess of maximum number of packets specified for the connection).
- 02 The message spent more than 15 seconds in transmission from the source Host to the IMP. This time is measured from the last bit of the leader through the last bit of the message.
- 03 The message was lost in the network due to IMP or circuit failures.
- 04 Resources unavailable.
- 05 Source IMP I/O failure during receipt of this message.

10 Interface Reset - The IMP's ready line has been dropped and pending output to the SIP has been discarded. This probably indicates that the SIP did not accept data from the IMP fast enough. Since dropping the ready line also sets the IMP's error flip-flop, the next message from the SIP will be discarded and answered with a Type 1 (Subtype 0) message.

15 New Format Message - This is a message in the new IMP-to-Host format. The IMP should never send these to the SIP; when it does, it is a mistake, and the SIP throws the message away.

The SIP also occasionally prints out messages concerning the status of the disk drives. A message is always printed out whenever the state of the drive controls is manually changed. These messages are of the form

THU16JAN75GMT03:45:17.2 DR UP(or DOWN)=xxxx

where xxxx is a number which is decoded according to the following table. Note that in this table, x stands for the drive number, either 0 or 1.

99x Nonexistent drive
100x Soft down
1x Write lock
10x Unsafe
2x Disabled
4x Offline
6x Unloaded (STANDBY)
20x Needed Volume Validation

Note that these numbers can be microcoded.

CONSOLE DECWRITER - INPUT

On type in, various control characters have special effects.

Control C stops the main SIP program and transfers control to STRUGL, provided toggle switch 12 is set. This is a drastic action that should seldom be used.

Control K issues a Reset to TENEX, provided toggle switches 10 and 12 are set.

Control L indicates the begining of an operator message to the CCP. Such a message should be terminated with a Control Z or aborted with a Control U. Messages can be more than one line long. Rubout can be used to back up one character of type in and Control R will retype the message so far. It is best to put a few short lines of asterisks before and after operator messages if you want to get the attention of the CCP people.

Control O stops any printout currently in progress.

Control Q starts a session of looking for and transmitting data to the Datacomputer. Such sessions are also started automatically if one is not already in progress at two minutes after every other hour.

The keyboard on the display terminal also has a couple of uses. Typing Control A followed by Control B aborts the SIP's waiting for any further Datalanguage if toggle switch 10 is set. Also, typing @M, @H, or @D while toggle switch 15 is set advances the SIP's clock by minutes, hours, or days, respectively.

RUNNING DIAGNOSTICS

Various DEC diagnostics can be run from the SIP. To run a diagnostic, first send an operator message to the CCP telling them that the SIP will be down for a while. Next, set toggle switches 10 and 12 and make sure that a Datacomputer session is not in progress. If one is, let it finish; the SIP will signal the end of the session with a DC QUIT system message. If you are in a hurry, hit ^K to abort the session. Next, hit Control C to bring STRUGL into memory, and hit Control U to stop its printout. When STRUGL prints its star, type a Control R followed by the name of the diagnostic and a carriage return; this will start the diagnostic. If STRUGL prints a question mark, this means it cannot find the diagnostic on either pack.

When finished, use the switches to restart STRUGL at location 157000. Reset all toggle switches to zero and give the command $$P to restart the SIP where it left off.

The SIP's packs should contain all the necessary diagnostics. To get a list of the diagnostics currently on the SIP's packs, give the command 0$V to STRUGL. Files numbered 10 and higher are diagnostics; they are stored by their DEC code names. The IMP diagnostic by BBN is called IMP11A.

If diagnostics are being run from a special diagnostic pack, the command 0^W should be given before the diagnostic is run. (Type a carriage return in response to the prompt "File Name".) To restart the SIP, use the hardware bootstrap which starts at address 773320, and then give the command 0^L to STRUGL, followed by the command $P.

CHANGING DISK PACKS

The SIP is currently able to store up to 42 hours of data on its two disk packs. Under normal circumstances, neither pack will become full. If the Datacomputer is down for an extended period of time, however, the packs may become full, and it will be necessary to change one of them to avoid losing data. No command is necessary to do this; the full pack may simply be switched with an empty pack. The only restriction is that only one pack be changed at a time. If a Datacomputer session is in progress when a pack is changed, the SIP will temporarily abort it so that the new Datacomputer session will use the directory from the new pack instead of continuing to use the old one. Whenever a pack is replaced with one in the current SIP format, the SIP automatically copies the system area from the unchanged pack onto the changed one. If the new pack is not in the current SIP format, the SIP first asks the user if he wants the pack initialized for the SIP. If the user answers no, the SIP unloads the pack and waits for another one.

Packs can also be changed while in STRUGL by means of STRUGL's ^P command, but care must be taken that the pack being changed is not the pack on which data is currently being stored, and that it is not STRUGL's system pack.

SIP CRASHES

Occasionally, the SIP will encounter a situation which it cannot handle, and it will crash. When it crashes, it will print out about a page and a half of somewhat useful information. (Hitting Control C will also cause it to stop and print out this information.) Generally, it is good to let all of this information be printed out. However, if the same error is occurring over and over again within the space of a few minutes, the printout can be interrupted by hitting any key on the DECwriter. If the printout ends without interruption, STRUGL will attempt to restart the SIP by swapping in a new copy of the SIP while saving the old SIP's directories; in this way, no data is lost. STRUGL will always try to restart the SIP in this manner unless the last crash was less than an hour ago, in which case STRUGL decides that the SIP is really sick and should be left alone, or unless toggle switch 13 was set by the time STRUGL got to the end of its printout. In both cases, STRUGL starts beeping plaintively once a second to let everyone know that the SIP is dead.

Repeated crashing of the SIP may be caused by one of two types of bugs. The SIP may either be getting a unique type of garbage from the CCP and/or TENEX that is causing previously hidden bugs to surface, or the SIP may have bashed its directories, which are retained during restarts. In the first case, not much can be done except to keep restarting the SIP with the 1^R command and hope that the offending host stops sending garbage soon. In the second case, the directories can be reinitialized with STRUGL's $$^Z command, and the SIP can then be restarted successfully with the 1^R command. This method has the disadvantage that all data stored on the disks to which the zeroed directories belong will be lost.

Whenever the SIP crashes and successfully restarts itself, a core image of the crashed SIP is automatically saved in file 0 (or file 100, if the second line of the bughalt printout says Drive 1). It is possible to retrieve and examine this file using STRUGL. First, stop the running SIP by hitting ^C and save it in an unused file. Next, use the hardware bootstrap (not the $^B or $$^B command) to bring a fresh copy of STRUGL into core, and give the 0^L command. The core image of the crashed SIP is now in user memory. To restart the running SIP, load it in using the ^L command and then give the $P command.

Occasionally, the SIP will not overtly crash, but will merely hang in some sort of loop. For example, something is drastically wrong if the display frame is not being constantly updated, or if the SIP's clock (line 1) is not running. In cases such as these, toggle switch 12 should be set and ^C should be hit to bring in STRUGL. When the printout finishes, hit a carriage return and give the 1^R command. If ^C does not work, use the switches to restart STRUGL at 157000, give the $X command, and continue as above.

If the SIP's drive 0 is dead, the 1^R command will not work, and the SIP must be restarted with the 101^R command.

APPENDIX B:

STRUGL User's Guide

by Steven A. Zimmerman

Computer Corporation of America

15 October 1977

INTRODUCTION

STRUGL is a debugger and micro-operating system which runs on the PDP-11/40 SIP; its name is an acronym for SIP's Total RUG Language. Although STRUGL was designed for the SIP, it will run as a fully functional debugging system on any PDP-11 processor of the 11/40 type or better that is equipped with the Extended Instruction Set (EIS). In addition to its debugging capabilities, STRUGL is able to write and load programs to and from disk, maintain directories of these programs, load from paper tape, downline load from the PDP-10, copy selected portions of either disk to the other, set up default trap handling for user programs, and act as a desk calculator.

Throughout this manual, an escape or altmode is represented by a dollar sign ($), while a control character is represented by an uparrow (^) preceding the character.

TABLE OF CONTENTS

HISTORY

STRUGL is a member of the RUG family with its prehistoric origins in the DEC ODT (octal debugging technigue) software. STRUGL traces its development from those humble beginnings through work done at the MIT Logo Project. At CCA the basic RUG program has served the Biocybernetics, SIP, and TDA projects as the principal debugging tool. For the three projects, RUG has been modified to produce five idiosyncratically different versions of RUG: GRUG (for the GT44), ARUG (for TDA), and FRUG, DRUG, and STRUGL (for the SIP). These versions are sufficiently different so that a guide to one is not, except in a general way, a useful guide to another. Also, STRUGL and ARUG are fundamentally different from the others because they are not co-resident in core with the user program being debugged as is DRUG, for example. Instead, a 1000 byte STRUGL resident portion, located from 157000-157777, alternately swaps the non-resident STRUGL portion and the user program being debugged between disk and core. The fact that only 1000 bytes of the STRUGL program need be resident in core eliminates the usual worries about the size of the debugger program and allows STRUGL to offer many more features than ODT, and also allows symbolic debugging with the full user program symbol table. STRUGL also goes a step further than ARUG in terms of swapping capabilities by having the entire breakpoint iteration routines contained in its resident part. Thus, to iterate through a breakpoint 1000 times it is no longer necessary to do 2000 program swaps. In fact, the iteration can be done in something closely approaching real time.

COMMAND INPUT

STRUGL's prompt is an asterisk (*); typing a carriage return (except as a quoted character on ASCII text input) should always cause STRUGL to reply with its prompt. If STRUGL dies during execution, restarting it at 157000 should restore its prompt. STRUGL comands are executed as soon as they are complete; there is no special character required to terminate the input of a command to STRUGL. Commands may be up to 80 characters long.

Expressions involving numbers, symbols, and the operators for add (+), subtract (-), multiply (*), divide (!), logical AND (^A), and logical OR (^O) are evaluated by STRUGL. Expressions are generally evaluated from left to right, with the exception that addition and subtraction are evaluated after all other operations. Parantheses may be used to impose grouping of operations. Note that on type-in numbers are assumed to be octal (even in $$D mode) unless suffixed by a period, in which case, they are assumed to be decimal numbers. Numbers which contain eights or nines are also assumed to be decimal numbers.

An ampersand causes the next three characters input to be packed as Radix-50 into one word. Single quote causes the ASCII value of the next character to be input; double quote causes the next two characters input to be packed into one word.

Delete (or rubout) deletes the last character typed in. Typing a ^U will delete the entire command string; a ^U will also abort any command currently in progress and restore STRUGL's prompt. Tabs are converted to three spaces on both input and output.

EXAMINING LOCATIONS

Types of Locations

In STRUGL, there are two basic types of locations: memory locations and registers. A memory location is represented by a number between 0 and 177777, while a register is represented by a per cent sign followed by a number between 0 and 77. Both types of locations can also be represented by symbolic names. (The locations .D, .RPCS1, .SR0, and .SS are referred to as registers throughout this manual, even though strictly speaking they are merely ordinary locations in STRUGL's resident page. However, they function in the same way as STRUGL's other registers.) Memory locations can refer to either the user program or to STRUGL itself, depending on the contents of the .S register. Locations below this value are interpreted as references to the user program; locations equal to or above this value are interpreted as references to STRUGL. Normally, .S is set to .H, which is 157000; this allows the entire user program plus STRUGL's resident portion to be accessed simultaneously. By setting .S to zero (its only other legal value), all of STRUGL can be accessed. When STRUGL prints out the contents of one of its own locations, it precedes the value with an exclamation point. The I/O page is considered to be part of STRUGL; locations on this page which may be modified by STRUGL are saved in special registers which are accessible to the user. Specifically, the user's program status word, stack limit register, memory management status register, and RP04 Status Register 1 are saved in STRUGL registers .PS, .SL, .SR0, and .RPCS1, respectively. The normal locations for these I/O registers contain temporary values used by STRUGL. The rest of the I/O page is not modified by STRUGL except by request. Programs which expect I/O registers to contain initial values must set up these registers with MOV instructions, not program origins.

Registers are merely special locations in STRUGL which are always accessible to the user, regardless of the setting of .S; the first eight hold the contents of the user's CPU registers, while the rest have assorted functions.

Opening a Location

One of the main uses of STRUGL is to examine locations in the user program being debugged. To examine a location, one "opens" the location by typing the address of the location followed by a slash (/) or backslash (\). A slash opens the location as a word; a backslash opens the location as a byte. The address of the location to be opened may be given as a number, a symbol, or an expression. For example, suppose that location 100 contains the value 1001, and suppose that the symbol BAR has the value 102. Then

```
*100/ 1001
*BAR-2/ 1001
*100\ 1
```

Examining Locations

```
*BAR-2\ 1
```

STRUGL types out the contents of the opened location; the type-out is according to the prevailing type-out mode. If a slash or a backslash is typed with no preceding argument, STRUGL reopens the current location.

An odd location cannot be opened in word mode; if the user types in an odd address followed by a slash, STRUGL echoes a backslash and forces the location open in byte mode.

Type-out Modes

There are five main type-out modes:

- C causes the contents to be typed as a number in the current radix
- S causes the contents to be typed as a symbol (plus numeric offset if necessary)
- I causes the contents to be typed as an instruction
- " causes the contents to be typed as a pair of ASCII characters
- [causes the contents to be typed as three radix 50 characters.

Note that all modes except I always type out the contents of one word or byte (depending on whether the location was opened in word or byte mode, i.e., with slash or backslash). However, in I mode the contents typed out may occupy one, two, or three words of memory depending on the instruction in the location opened. Note also that in S, I, or [mode, bytes are nevertheless typed out in C mode. If while in I mode a word cannot be typed out as an instruction, an attempt is made to type it out as a symbol. EMT and TRAP instructions are printed out as both instructions and symbols when in I mode, since they are usually referred to by user-assigned symbolic names.

There are four subsidiary modes:

- O sets the radix to octal (default)
- D sets the radix to decimal
- A shows address fields in instructions as absolute numbers
- R shows address fields in instructions as relative addresses (symbol plus offset) (default)

There are two modes governing the printout of octal numbers:

- M prints all octal numbers as negative numbers
- ^P prints all octal numbers as positive, unsigned numbers (default)

At any given time, there is a prevailing type-out mode. The user may change the prevailing type-out mode temporarily (i.e., until STRUGL types its asterisk, at which point the mode reverts to the previously prevailing mode) or "permanently" (i.e., until he changes

the mode "permanently" again). The command to change the mode temporarily is escape followed by the mode letter. To change the mode "permanently" it is two escapes followed by the mode letter. If the command is preceded by an argument, that argument is retyped in the new mode; otherwise, the last value typed by STRUGL is retyped in the new mode.

There are two subtypes for the instruction mode type-out. The first one is the normal assembler syntax, which is the default. The second one simulates the .ABS directive of PALX. When this mode is enabled, instructions such as CLR @#VAL are output as CLR VAL, and instructions which are input in the form CLR VAL are assembled as if they were input CLR @#VAL. Instructions which occur in real relative format are output in the form CLR *VAL to differentiate them from absolute format instructions. The .ABS mode is enabled by the $^A command and disabled by the $$^A command; both modes are "permanent" modes.

The ^C command is a combination of the $C and slash commands; the ^S command is a combination of the $S and slash commands.

When a value is typed out, it may be viewed in other modes by means of the equals command. (The equals command may also have arguments of its own.) The first time the equals key (or backarrow or underbar key) is hit, the value is typed as a number in the current radix. If the equals command is given again, the value is typed in the other radix; repeated equals commands then produce the value typed as a negative octal number (or positive, if the first one was negative), a symbol, an instruction, two ASCII characters, and three Radix-50 characters. Typing another command after an equals command recycles the equals command back to its original numeric printout.

Modifying the Contents of the Open Location

It is of course not enough to see mistakes by having the contents of some location typed out; it is necessary to correct them. When a location is open, its contents can be changed. When a location is open, user type-in except for commands to STRUGL is evaluated and used to replace the previous contents of the location. The type-in can be a number, a symbol, an instruction, or an expression involving some or all of the three. Note that typing in an instruction may modify more than the word currently open; it can possibly modify the next word or the next two words in memory depending on the instruction typed in. To type in an instruction follow the normal assembly language format. Extra spaces or tabs between the operation code and any arguments are ignored. STRUGL recognizes all PDP-11 instructions except the CLx and SEx (clear/set condition code) instructions. These must be typed in specially as 'CL' or 'SE' followed by a space (or tab) and then the condition code character (one of 'C', 'V', 'Z', 'N'); more than one condition code may be specified and each should be separated from the preceeding one by a space.

Suppose that FOO is a defined symbol and that location 100 contains the value 53. The following are all acceptable:

```
*100/  53  MOV #13.,@FOO+.  (note 13 decimal)
*100/  53  FOO+36-400
*100/  53  6
*100/  53  CL Z V
```

Closing an Opened Location

A carriage return closes the currently open location after replacing its previous contents with the user modification, if any. A carriage return returns to STRUGL's top-level command decoder and types the STRUGL prompt, asterisk. The following STRUGL commands in addition to carriage return also close the currently open location after making any modifications specified by the user: linefeed, uparrow, question mark, right angle bracket, and left angle bracket.

Opening Locations Related to the Current Location

STRUGL has several commands to open "related" locations.

Linefeed will open the next sequential location in memory. In byte mode, it opens the byte at location+1. In all other modes except I mode, linefeed opens location+2. In I mode linefeed opens the next instruction at location+2, location+4, or location+6 depending on the instruction in location (i.e., it skips over the other words of a multiple-word instruction).

Uparrow will open location-1 in byte mode and location-2 in word mode. Note that this is independent of mode (I mode included). In I mode, uparrow will often give misleading type-out; it will type out the contents of location-2 as an instruction, although it is not necessarily the instruction preceeding the instruction in the program. Since there is no way for STRUGL to know on which word the preceding instruction began, there is no way to avoid this problem.

The $U command performs as many linefeeds in a row as is specified in its argument. If no argument is specified, the command will perform successive linefeeds until ^U is typed.

The $$U command is like the $U command except that it performs successive uparrow commands instead of successive linefeeds.

Right angle bracket opens the location mentioned in the currently open location. For example, if the open location contains

```
    MOV BAR,FOO
    JMP FOO
or  TSTB @FOO
```

then right angle bracket will open FOO.

Question mark opens the next-to-last location mentioned in the currently open location. For example, if the open location contains

```
MOV BAR,FOO
```

then question mark opens BAR. If there is only one location mentioned, then question mark is the same as right angle bracket.

Right angle bracket and question mark when typed after the

contents of the currently open location have been typed out both save the address of the currently open location before opening the related location. These addresses are placed in an eight-cell ring buffer. The command left angle bracket returns to the previously open location and reopens it after first closing the related location. Thus it is possible to chase down up to eight levels of subroutine calls, for example, by typing a right angle bracket after each JSR PC,xxxx is typed, and then after the exploration is complete, to return to the originally opened locations by successive left angle bracket commands.

BREAKPOINTS

To examine the state of his program, the user can set breakpoints in the program, which cause control to revert to STRUGL when execution of the user program reaches the point at which a breakpoint is set. Control will also revert to STRUGL whenever the BPT instruction is executed in the user's program or upon execution in the user's program of any instruction with the T-bit set in the PS. It is assumed that the user program does not alter the contents of the breakpoint trap vector (locations 14 and 16).

Setting Breakpoints

To set a breakpoint at an address, type the address followed by $B. This sets the next available breakpoint; STRUGL allows the setting of up to eight simultaneous breakpoints, which are identified as B0-B7. Breakpoints are set in either kernel, user, or absolute address space, depending on the setting of the .MM register.

Breakpoints are not actually inserted into the user program until an $G, $P, or ^N command is given. For this reason, the instruction at a breakpoint may be examined and even modified after the breakpoint is set.

Breakpoints may be set in STRUGL itself by setting .S to zero before issuing the $B command. Breakpoints which are set in STRUGL are inserted immediately. Breakpoints may be set in either the user program or in STRUGL, but not in both simultaneously. The value of .S must not be changed while any breakpoints remain set.

Reaching a Breakpoint

When a breakpoint location is reached by program flow in the user program, the execution of the BPT instruction causes a trap to STRUGL, which swaps out the user program and swaps in the STRUGL non-resident portion. It then types out a message saying which breakpoint was reached and the address and contents of the location reached; the message has the form:

Bn; Oaddress Saddress/ instruction

where n is 0-7 or E if the BPT executed was not one set by a command to STRUGL, Oaddress is the address of the breakpoint in octal, and Saddress is the symbolic address of the breakpoint. After typing out this message, STRUGL will prompt the user with an asterisk. The user can execute any STRUGL command at this point. The $X command will give an extended breakpoint type-out, printing STRUGL's current system drive, the current setting of the switch register, the values of R0-R5, SP, the entire stack, the user mode SP, the entire user mode stack, the .PS and .SR0 registers, and the contents of all the Active Page Registers if memory management is present. The $X command is executed automatically if the breakpoint was a SIP breakpoint.

Proceeding from a Breakpoint

The user may continue execution of his program by typing $P. Optionally, the escape may be preceded by a number, namely the number of times to proceed through this breakpoint again without breaking to STRUGL. Note that a proceed count is legal only for breakpoints which were set by STRUGL.

Removing Breakpoints

The command $$B removes all breakpoints. An address followed by $$B removes a breakpoint at that address. Alternatively, $B followed by a number 0-7 will remove the breakpoint specified by its number.

Breakpoint Registers

Opening the register .B+n where n is a breakpoint number 0-7 will show the address at which that breakpoint is set (or -1 if not set). Opening the register .C+n will show the proceed count for that breakpoint, usually 1. By setting this count to a value v before typing $G, the user program will loop through the breakpoint v times before making the initial break to STRUGL. Opening the register .L+n shows the address of a location whose contents should be typed out after the breakpoint message for that breakpoint. If this location contains -1, no special type out is made.

Single-Step Mode

As an alternative to proceeding from a breakpoint with the $P command, the user may proceed in single-step mode by typing ^N optionally preceded by the number of instructions to execute in single-step mode. (The default number is one.) After executing some number of instructions in single-step mode, the user can proceed with the $P command. After execution of each instruction in single-step mode, STRUGL prints a message like the breakpoint message except that 'Bn' is replaced by 'SS'. This message indicates the location reached by the execution of the prior instruction.

SEARCHES

The user can search a bounded portion of memory for a particular value. The search registers are .M, .M+1, and .M+2, which contain the mask, low search limit, and high search limit respectively. Normally their values are -1, Ø, and 157776, but they may be set by the user to suit his needs. The mask is ANDed with each word of memory in the search range before being tested by the search. There are three kinds of searches: word, effective address, and not, which corresponds to the commands $W, $E, and $N. Each is preceded by the value sought. Word search lists all locations having the value sought, not search lists all locations not having the value sought, and effective address search lists all locations whose effective address is the value sought. Typing ^U aborts a search.

THE SYMBOL TABLE

STRUGL has many predefined symbols, including symbols for the PDP-11 CPU registers (R0, R1, R2, R3, R4, R5, SP, and PC), symbols for many of the more commonly used registers on the I/O page, and symbols for many device interrupt vectors. (The last are all half-killed.) There is room for over 4000 user defined symbols in STRUGL. The register ..B contains the current lower limit of the symbol table; this limit should be changed manually only with great care.

Defining Symbols

Normally the user program's symbol table is available to STRUGL. The user may extend this symbol table by adding new symbols or by redefining old ones. To define a symbol, type

```
value,symbol:  for example,
100,FOO:   defines FOO to have the value 100.
```

A previously undefined symbol can be defined to have the value of the currently open location by typing simply symbol: (i.e., the value is implicit). A symbol can be "half-killed", that is flagged so that STRUGL will not use it for type-out but will still recognize it for input, by typing the symbol followed by ^K. To unhalf-kill a symbol, merely type

```
symbol,symbol:
```

Note that if a symbol occurs in the symbol table more than once, then it is impossible to half-kill any but the first occurrence of it.

MEMORY MANAGEMENT

STRUGL has complete facilities for utilizing memory management. Normally, when a program is loaded with the $LT command, it is loaded into the physical addresses specified by its origins. If the program uses memory management, its origins specify virtual addresses, and these must be converted to physical addresses before the program can be loaded. The method STRUGL uses to determine how to perform this conversion is to require that all programs which use virtual addressing for their code have the first origin of the program set to 360; this origin should be followed by the contents of the eight page address registers (either kernel, user, or both) which are utilized by the program. Note that there is an inherent limit of 32K on the size of any one program; programs bigger than this limit must be broken up into overlays (see "Downline Loading").

When a program is residing in memory, the contents of the .MM register determine the memory management mode used for calculating physical addresses. When .MM contains a zero, all input addresses are considered to be physical addresses. When .MM contains a 1 or a -1, all input addresses are considered to be virtual addresses which refer to kernel or user space, respectively, and the appropriate page address registers are used to construct physical addresses. When the contents of locations in kernel or user space are printed out, they are preceded by a *K'* or *U'*. Setting the .S register to zero overrides the .MM register.

THE FILE SYSTEM

STRUGL is equipped with a file system which allows user programs to be read from and written onto either disk. Files are numbered from 0 to 36 (octal); file 0 is reserved for system use and files 1 through 36 are available to the user. Files 0 through 5 are double sized files and can hold programs as big as 64K words; files 6 through 36 can hold programs as big as 28K words. In a SIP system, file 1 is reserved for the current version of the SIP. Files on disk drive 1 are numbered from 100 to 136, and are functionally equivalent to those on drive 0.

The four main file commands are ^L (load a file from disk), ^W (write a file to disk), ^R (run a file), and ^D (delete a file). (The ^R command is actually a combination of the ^L and $G commands.) These four commands can either refer to files by number, in which case the number precedes the command, or by name, in which case the name directly follows the command. If more than one file has the same name, a reference by name will cause STRUGL to choose the lowest numbered file with that name. Note that STRUGL does command name completion for these four commands.

The ^W command always requests a name for the file being written, even if one was just specified; this is so the file's name can be easily changed. A file name of up to eight characters should be entered; if the first character is a carriage return, the previous name is retained.

The current state of STRUGL's registers is saved with each program when the ^W command is executed, and the registers are restored to this state whenever the program is loaded with the ^L or ^R commands. The contents of the memory management registers are also saved and restored with each program.

Starting the User Program

To start the user program, type the start address followed by $G. If the user program was assembled with a "jump address", the start address may be omitted. In this case, STRUGL will perform a complete system initialization before starting the user program, including issuing a hardware RESET instruction. The register .G contains the default start address, which may be changed by the user. If .G contains a 1, there is no default start address.

Whenever a program is loaded into user memory, the stack pointer is initialized to 1000, and the program starting address and initial program status word are temporarily stored in the previous two locations when the program is started. If the user program contains instructions or data in locations 774 and 776, the stack pointer should be set to a different value before giving the $G command.

Directories

STRUGL's two disk directories can be printed out with the $V command. If no argument is given, both directories will be printed.

The commands 0$V and 100$V print the drive 0 and drive 1 directories, respectively. To print the directory entry for a single file, type the file number followed by $V. The entry for each file contains the file number, the file name, the version of STRUGL that wrote the file, the relative order in which the file was last written, and the date the file was last written.

The command ^V types the current STRUGL version number followed by the numbers of the files last read and written.

Directories may be zeroed by means of the $^Z command which requires the directory number (0 or 1) as its argument. Upon typing $^Z, STRUGL will type "Zero STRUGL directory?" The command should be confirmed with the letter Y; any other character will abort it. The $^Z command must be used on disk packs before they can be used by STRUGL, or if STRUGL's file structure is changed. If files have been stored on a pack which has not been zeroed, they should be copied to a spare pack, the first pack should be zeroed, and the files should then be copied back. The one instance where the $^Z command need not be given is where a new disk pack is being copied from an old one using the $^C command.

Copying Disks

The system area and files of one disk may be copied to another by means of the $^C command. The command 0$^C copies from drive 0 to drive 1, while the 1$^C command copies from drive 1 to drive 0. The $^C command with no argument is identical to 0$^C. The various forms of the $$^C command are equivalent to those of the $^C command, except that they copy the system area only. An automatic $$^C to the non-system disk is performed by STRUGL whenever an $G command is given with no argument, or whenever a ^R command is given. Whenever the ^P command is given, STRUGL performs an automatic $$^C to the new pack. As a result, there is little need for the user to ever directly execute this command.

DOWNLINE LOADING

Loading Programs from Paper Tapes

The $LP command allows paper tapes in standard binary format to be loaded to STRUGL. This command uses the high speed reader; the command Ø$LP uses the low speed reader. Both of these commands zero user core and set up STRUGL's trap vectors before loading a program; the commands $^LP and Ø$^LP do not. If a program exists on multiple paper tapes, STRUGL will halt at the end of each one to allow the user to load the next one and press Continue. When a load has been completed successfully, STRUGL types its prompt.

Loading Programs from the PDP-1Ø

The command $LT causes further input to be transmitted to the PDP-1Ø and output from the PDP-1Ø to be echoed on the STRUGL console terminal. STRUGL's terminal appears to the user at this point to be a standard TENEX terminal. The user should log in and may execute any legal TENEX commands. To actually load to STRUGL, the program NLDSIP in the SIP directory should be run. NLDSIP asks four questions. If the current answer is terminated with a comma, the next question is asked; if it is terminated with a carriage return, default answers are used for the remaining questions. The four questions are "INPUT FILE:", "LOAD SYMBOLS(Y/N)?", "ARE YOU THE PDP-11 (Y/N)?", and "AUTO-LOGOUT WHEN DONE (Y/N)?"; the default answers to the last three are Y, Y, and N. For further information on NLDSIP, see the NLDSIP Manual.

When the program is being loaded, STRUGL types a G for every good binary block received, a J for the jump block following the binary blocks, an S for every good symbol block received, and a Z for the jump block following the symbol blocks. STRUGL types a B when it receives a block with a bad checksum and tells NLDSIP to resend that block. If STRUGL appears to hang for a long time during the loading process, data has probably been lost between the PDP-1Ø and the SIP; typing ^F will send a bad block acknowledge to NLDSIP and cause it to resend the current block.

The $LT command zeroes user core and sets up STRUGL's trap vectors before loading programs; the $^LT command does not. Typing ^U at any time will return to the STRUGL command decoder.

A program can be built up from a number of overlays by issuing repeated calls to NLDSIP while STRUGL is connected to TENEX. If this procedure is being followed, it is important that ^U not be hit until all overlays have been loaded.

All registers are reset to their initial values any time any of the forms of the $L command are executed.

BOOTSTRAPS

Whenever a new version of STRUGL is started up with the $G command, it saves a spare copy of itself in a special system area and writes a bootstrap to sector 0 of the system disk. If the version of STRUGL currently being run ever seems partially bashed, the $^B command will boot in this spare copy, leaving the symbol table and registers intact. If $^B is not sufficient to restore a running STRUGL, $$^B will boot in the spare STRUGL with an initialized symbol table and initialized registers. The disadvantage of this command is that the current user program is effectively lost.

If not even a partially working STRUGL exists in memory, the hardware bootstrap should be used. Set the switches to 773322, press Load Address, set the switches to zero, and press Start. This should boot in and start a fresh copy of STRUGL. To boot in the current swapping STRUGL, set switch 8 before pressing Start. To boot STRUGL in from drive 1, set switch 0 before pressing Start.

Generally, if the situation is bad enough that the hardware bootstrap is needed, the current swapping STRUGL is probably not reliable, and the spare, fresh copy should be the one booted in.

Occasionally, the program running in core may bash the resident portion of STRUGL and then either die itself or attempt to execute a breakpoint. In these cases, it may be useful to save the remains of the user program before reading STRUGL into core. This can be accomplished by using the hardware bootstrap and setting switch 11 before pressing Start. This reads in a fresh copy of STRUGL's resident portion, saves the rest of core in the user program area on disk, and reads in the rest of the current swapping STRUGL. Switch 8 has no effect when using this bootstrap, but switch 0 still determines the drive used. Note that the first 2000 bytes of the program are lost by this method because they are overlaid with the bootstrap.

If patches are made to the running STRUGL, this patched version can be saved as the spare copy with the $^W command. If the bootstrap is changed, a new bootstrap can be written to disk by giving the ^B command preceded by the number of the drive to which it is to be written; ^B with no argument will write the bootstrap to both drives.

ERROR HANDLING ROUTINES

Traps

STRUGL sets up default trap vectors for programs loaded with the $LP, $LT, and $L^T commands. When STRUGL services a user program trap (or even a STRUGL trap), it prints out a trap message similar to the breakpoint message except that the "Bn;" is replaced by a phrase explaining what kind of trap occurred.

If the trap is a reserved instruction trap, STRUGL usually will not be able to find a mnemonic for the instruction which caused the trap. In this case, STRUGL will print the instruction in octal. For all other traps, if the word the PC is pointing at does not look like an instruction, STRUGL assumes that the trap occurred in the middle of a two or three word instruction; in this case, it backs the PC up by two and tries again. When it does this, it prints "-2/" to let the user know that the instruction printed out actually precedes the address of the trap.

Disk Errors

A disk error will abort the current operation and cause STRUGL to print out the message "Disk Error on Drive x" followed by the current contents of disk registers RPCS1, RPCS2, RPER1, RPER2, and RPER3. If the error is a write lock error, STRUGL's message is "Write Lock Disk Error on Drive x".

COMMANDS FOR THE SIP

Loading New SIPs

When a new SIP is loaded from the PDP-10, STRUGL normally grafts the directories of the old SIP onto the new one automatically, in order to avoid losing any data. In cases where the new SIP directory structure is different from the old one, this obviously must be avoided. For these cases, the $L^T command should be given in place of the $LT command. In addition to preserving the new directories, the $L^T command saves an initialized directory in a special area on the system disk; this area is referred to by the $$^Z command (see below).

Changing Disk Packs

Whenever one of the SIP's disk packs is changed, its directory must be written onto it, and the directory of the new pack must be read in in its place. This is all done automatically by the ^P command. When a disk pack is to be changed, the ^P command should be given, preceded by the disk's drive number. Some version of the SIP must be in user memory at this time. STRUGL will power the drive down by putting it on STANDBY; the packs should be switched and STANDBY hit to power the drive back up. It is not necessary to ever hit the START/STOP switch. Normally, STRUGL's system disk should not be switched by this procedure, as it is impossible to continue the running SIP from a different system disk.

Disk packs can be changed when the SIP is actually running without the need for any command. This method is better than STRUGL's ^P command because either pack can be changed this way.

Reading In a SIP Directory

Occasionally, the user may want to replace one of the SIP's current directories with a directory stored on one of the disk packs, without actually changing disk packs. The $^D command exists for this purpose. Typing the number of the drive the disk is on followed by $^D will read into user memory the directory for that disk. A copy of the directory is also stored in the proper place in file 0. Some version of the SIP must be in user memory for this command to work.

Reloading the Last Running SIP

At various times the running SIP must be interrupted to run diagnostics, load programs from the PDP-10, patch programs, etc. The $^R and $$P commands allow the user to reload and proceed from the running SIP when these procedures are finished. Whenever a program is loaded into memory, either from the disk, paper tape, or the PDP-10, or when user memory is zeroed, STRUGL firsts checks to see whether the program currently in memory is a SIP which had actually been running at some point. If it is, STRUGL automatically

saves it in file 0. It may be brought back into user core at any time by the $^R command; the $$P command is equivalent to an $^R command followed by an $P command. Note that the $L^T command intentionally deletes any running SIP in file 0.

Zeroing a SIP Directory

The SIP has two directories, numbered 0 and 1, which keep track of the data stored on the correspondingly numbered drive. The $$^Z command preceded by a directory number will replace the specified directory with an initialized, zeroed directory. The pack ID of the directory is not affected. Some version of the SIP must be in user memory for this command to work.

The Saved SIP Register

Whenever a program is loaded into user memory immediately after a version of the SIP has been running, the SIP is first automatically transfered from user memory into file 0, and STRUGL's .SS register is incremented. Whenever a copy of the SIP is loaded into user memory and the .SS register is nonzero, STRUGL automatically retrieves the SIP directories from file 0 and inserts them into the newly loaded SIP. This way, the SIP in user memory always contains the most recent copy of the directories. The $L^T command clears the .SS register.

At times it may be useful for the user to manually set the .SS register. For example, any time the hardware bootstrap is used, the fresh copy of STRUGL starts out with .SS set to zero. If the user wants to be able to use the directories in file 0, .SS should be set to 1. On the other hand, any time the user wants to use the directories in a saved copy of the SIP just as they are and not have the current directories loaded over them, .SS should be set to zero before the ^L command is given. If a previously run SIP is in user memory at this time, a random file should be loaded into memory before modifying the .SS register.

Printing the SIP's Status Frame

Typing $F will cause STRUGL to print the SIP's status frame as it was when the SIP last stopped running. Note that this frame is always slightly more up to date than the one on the display. The last running SIP must be in user memory for this command to work.

MISCELLANEOUS COMMANDS

Initializing Memory

The $Z command reinitializes the symbol table to be empty except for the STRUGL predefined symbols. The $$Z command zeroes all core, reinitializes the symbol table, and sets up STRUGL's default trap vectors. The value of the symbol ..B is the lower limit of the symbol table; the least allowed value for ..B is 44000.

Comments

STRUGL interprets any command beginning with a semicolon to be a comment, and merely echoes all incoming characters up until the next carriage return.

Disk Read Test

The $K command will successively read in the first 65,536 disk sectors on the system disk into a spare buffer; this is about 1/3 of the disk. The routine will abort an any errors.

MISCELLANEOUS REGISTERS

.

Dot is STRUGL's current location counter.

.D

This register contains the number of STRUGL's current system drive. In general, it should be modified by the user only directly after an $^C or $$^C command.

.O

This is the disk offset register, which is normally set to zero. STRUGL adds the contents of this register to all disk addresses before reading from or writing to the disk. By making use of the fact that the special symbol .SW is equal to the starting disk block of the swapped user program, it is possible to examine the first 64K disk blocks on either disk. Since the command y/ examines location y in disk block .SW, setting .O to x-.SW will allow the command y/ to examine location y in disk block x. Remember to reset .O to zero when finished, or STRUGL will die horribly.

.P

This register initially contains the priority at which STRUGL runs. Changing this number does not affect anything.

.V

This register contains the last value typed by STRUGL.

.W

This register specifies the width of the symbol offset tolerance. Symbolic addresses and instruction operands are printed in the form symbol+offset as long as offset is less than or equal to the contents of the .W register; otherwise, they are printed as numbers. When type-out is occurring in the S mode, however, this restriction on offset size is temporarily suspended.

PREDEFINED SYMBOLS

The following symbols always exist in STRUGL's symbol table. All of STRUGL's registers are also predefined symbols.

Symbol	Value	Meaning
.H	157000	High limit of user program
.SW	51	Starting sector of swapped user program
..L	124000	Low address of actual STRUGL program
%PS	177776	Program status word
%SWR	177570	Console switch register
%TKS	177560	Teletype/DECwriter registers
%TKB	177562	
%TPS	177564	
%TPB	177566	
%TKV	60	
%TPV	62	
%PRS	177550	Paper tape reader and punch registers
%PRB	177552	
%PPS	177554	
%PPB	177556	
%PRV	70	
%PPV	72	
%PKCSR	172540	Real time clock registers
%PKCSB	172542	
%PKC	172544	
%PKV	104	
%LKS	177546	Line clock register
%LKV	100	

COMMAND SUMMARY

Command	Page	Description
^B	19	Write bootstrap to disk
^C	8	Equivalent to $C followed by slash
^D	16	Delete a file
^K	14	Half-kill a symbol
^L	16	Load a program from disk
^N	12	Single step
^P	21	Change disk packs
^R	16	Run a program
^S	8	Equivalent to $S followed by slash
^V	17	Type STRUGL version number
^W	16	Write a program to disk
=	8	Print current value in successive modes
:	14	Define a symbol
/	6	Open a word
\	6	Open a byte
[CR]	9	Close the current location
[LF]	9	Close the current location and open the next one
?	9	Open the first address mentioned
>	9	Open the referenced location
<	9	Return to the previous sequence and open
^	9	Open the previous location
;	23	Insert a comment
$A	7	Temporarily inhibit symbolic address type-out
$B	11	Set or delete a breakpoint
$C	7	Set temporary constant mode
$D	7	Set temporary decimal mode
$E	13	Search for effective address
$F	22	Print the SIP's status frame
$G	16	Start the user program
$I	7	Set temporary instruction mode
$K	23	Disk read test
$L	18	Zero core and load from the PDP-10 or paper tape
$M	7	Set temporary negative octal mode
$N	13	Do a "not" search
$O	7	Set temporary octal mode
$P	12	Proceed from a breakpoint
$R	7	Temporarily re-enable symbolic address type-out
$S	7	Set temporary symbolic mode
$U	9	Do successive line feeds to the terminal
$V	16	Type out STRUGL's directories
$W	13	Search for the specified word
$X	11	Do extended breakpoint type-out
$Z	23	Zero the symbol table
$^A	8	Type out instructions in absolute format
$^B	19	Boot in the spare STRUGL from disk
$^C	17	Copy the disk system area and files
$^D	21	Read in a SIP directory
$^L	18	Load from the PDP-10 or paper tape

$^P	7	Set temporary positive octal mode
$^R	21	Reload the last running SIP
$^W	19	Save the current STRUGL as the new spare STRUGL
$^Z	17	Zero a STRUGL directory
$"	7	Set temporary ASCII mode
$[	7	Set temporary Radix-50 mode
$$A	7	Permanently inhibit symbolic address type-out
$$B	12	Delete breakpoints
$$C	7	Set permanent constant mode
$$D	7	Set permanent decimal mode
$$I	7	Set permanent instruction mode
$$M	7	Set permanent negative octal mode
$$O	7	Set permanent octal mode
$$P	21	Proceed to the previously running SIP
$$R	7	Permanently re-enable symbolic address type-out
$$S	7	Set permanent symbolic mode
$$U	9	Do successive uparrow commands to the terminal
$$Z	23	Zero user memory and the symbol table
$$^A	8	Disable $^A command
$$^B	19	Do $^B command, then initialize the symbol table
$$^C	17	Copy the disk system area
$$^P	7	Set permanent positive octal mode
$$^Z	22	Zero a SIP directory
$$"	7	Set permanent ASCII mode
$$[	7	Set permanent Radix-50 mode

REGISTER SUMMARY

Register	Page	Description
R0	6	User R0
R1	6	User R1
R2	6	User R2
R3	6	User R3
R4	6	User R4
R5	6	User R5
SP	6	User SP
PC	6	User PC at last breakpoint or trap
.	24	Current location counter
.B	12	Breakpoint address registers
.C	12	Breakpoint count registers
.D	24	Contains number of STRUGL's current system drive
.G	16	Contains user program default starting address
.L	12	Contains location to be dumped at breakpoints
.M	13	Contains mask for searches
.M+1	13	Contains lower limit of search
.M+2	13	Contains upper limit of search
.MM	15	Memory management switch
.O	24	Contains offset to all disk addresses
.P	24	Contains STRUGL's priority
.PS	6	User program status word
.RPCS1	6	User RPCS1
.S	6	Contains limit for STRUGL self-examines
.SL	6	User stack limit register
.SR0	6	User memory management status register 0
.SS	22	Nonzero if a valid SIP exists in file 0
.V	24	Contains last value typed by STRUGL
.W	24	Contains width of symbol offset tolerance
..B	14	Contains the lower limit of the symbol table